# 消息队列的优点与缺点

优点：解耦、异步、削峰

缺点：1、系统可用性降低：系统引入的外部依赖越多，越容易挂掉，当MQ挂的时候，会导致整套系统全部崩溃无法使用。

2、系统的复杂性提高：加入MQ，如何保证消息有没有被重复消费？怎么处理消息丢失的情况？怎么保证消息传递的顺序性？

3、一致性问题：系统A处理完成后直接给用户返回成功，所以在用户的角度就任务系统处理成功，但是系统A还需要调用系统BCD等进行处理，如果后续系统中有一个系统处理失败，则会产生数据不一致问题。

# kafka、activemq、rabbitmq、rocketmq都有什么优缺点？

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 特性 | ActiveMQ | RabbitMQ | RocketMQ | Kafka |
| 单机吞吐量 | 万级，吞吐量比RocketMQ和Kafka要低了一个数量级 | 万级，吞吐量比RocketMQ和Kafka要低了一个数量级 | 10万级，RocketMQ也是可以支撑高吞吐的一种MQ | 10万级别，这是kafka最大的优点，就是吞吐量高。    一般配合大数据类的系统来进行实时数据计算、日志采集等场景 |
| topic数量对吞吐量的影响 |  |  | topic可以达到几百，几千个的级别，吞吐量会有较小幅度的下降    这是RocketMQ的一大优势，在同等机器下，可以支撑大量的topic | topic从几十个到几百个的时候，吞吐量会大幅度下降    所以在同等机器下，kafka尽量保证topic数量不要过多。如果要支撑大规模topic，需要增加更多的机器资源 |
| 时效性 | ms级 | 微秒级，这是rabbitmq的一大特点，延迟是最低的 | ms级 | 延迟在ms级以内 |
| 可用性 | 高，基于主从架构实现高可用性 | 高，基于主从架构实现高可用性 | 非常高，分布式架构 | 非常高，kafka是分布式的，一个数据多个副本，少数机器宕机，不会丢失数据，不会导致不可用 |
| 消息可靠性 | 有较低的概率丢失数据 |  | 经过参数优化配置，可以做到0丢失 | 经过参数优化配置，消息可以做到0丢失 |
| 功能支持 | MQ领域的功能极其完备 | 基于erlang开发，所以并发能力很强，性能极其好，延时很低 | MQ功能较为完善，还是分布式的，扩展性好 | 功能较为简单，主要支持简单的MQ功能，在大数据领域的实时计算以及日志采集被大规模使用，是事实上的标准 |
| 优劣势总结 | 非常成熟，功能强大，在业内大量的公司以及项目中都有应用    偶尔会有较低概率丢失消息    而且现在社区以及国内应用都越来越少，官方社区现在对ActiveMQ 5.x维护越来越少，几个月才发布一个版本    而且确实主要是基于解耦和异步来用的，较少在大规模吞吐的场景中使用 | erlang语言开发，性能极其好，延时很低；  吞吐量到万级，MQ功能比较完备  而且开源提供的管理界面非常棒，用起来很好用  社区相对比较活跃，几乎每个月都发布几个版本分  在国内一些互联网公司近几年用rabbitmq也比较多一些  但是问题也是显而易见的，RabbitMQ确实吞吐量会低一些，这是因为他做的实现机制比较重。  而且erlang开发，国内有几个公司有实力做erlang源码级别的研究和定制？如果说你没这个实力的话，确实偶尔会有一些问题，你很难去看懂源码，你公司对这个东西的掌控很弱，基本职能依赖于开源社区的快速维护和修复bug。  而且rabbitmq集群动态扩展会很麻烦，不过这个我觉得还好。其实主要是erlang语言本身带来的问题。很难读源码，很难定制和掌控。 | 接口简单易用，而且毕竟在阿里大规模应用过，有阿里品牌保障  日处理消息上百亿之多，可以做到大规模吞吐，性能也非常好，分布式扩展也很方便，社区维护还可以，可靠性和可用性都是ok的，还可以支撑大规模的topic数量，支持复杂MQ业务场景  而且一个很大的优势在于，阿里出品都是java系的，我们可以自己阅读源码，定制自己公司的MQ，可以掌控  社区活跃度相对较为一般，不过也还可以，文档相对来说简单一些，然后接口这块不是按照标准JMS规范走的有些系统要迁移需要修改大量代码  还有就是阿里出台的技术，你得做好这个技术万一被抛弃，社区黄掉的风险，那如果你们公司有技术实力我觉得用RocketMQ挺好的 | kafka的特点其实很明显，就是仅仅提供较少的核心功能，但是提供超高的吞吐量，ms级的延迟，极高的可用性以及可靠性，而且分布式可以任意扩展  同时kafka最好是支撑较少的topic数量即可，保证其超高吞吐量  而且kafka唯一的一点劣势是有可能消息重复消费，那么对数据准确性会造成极其轻微的影响，在大数据领域中以及日志采集中，这点轻微影响可以忽略  这个特性天然适合大数据实时计算以及日志收集 |

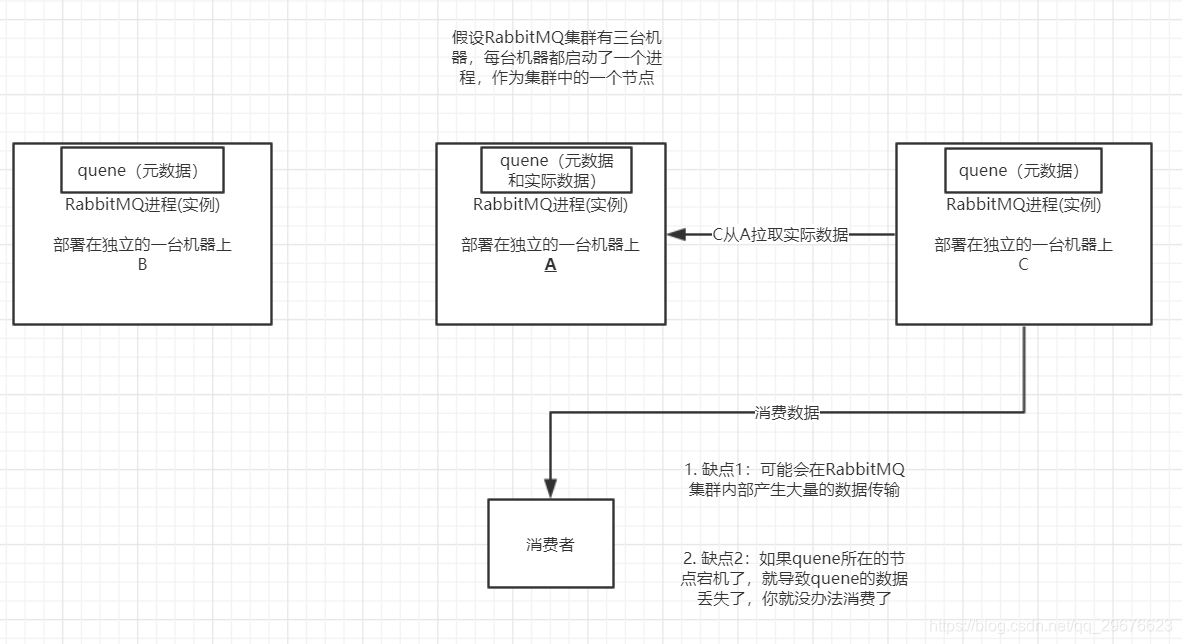
# 引入消息队列之后如何保证其高可用性?

（1）RabbitMQ的高可用性

RabbitMQ是比较有代表性的，因为是基于主从做高可用性的，我们就以他为例子讲解第一种MQ的高可用性怎么实现。

rabbitmq有三种模式：单机模式，普通集群模式，镜像集群模式。

**普通集群模式**：就是消费数据的时候都会从实际存放数据的queue中拉取数据，导致的结果就是网络获取的开销很大，并且当实际存储的queue所在的节点挂了，那么可能产生数据丢失，消费不到数据（因为这导致你要么消费者每次随机连接一个实例然后拉取数据，要么固定连接那个queue所在实例消费数据，前者有数据拉取的开销，后者导致单实例性能瓶颈）。如下图：



**镜像集群模式**：开启应有策略，把queue的数据同步到其他queue上，带来的缺点是，当某个queue数据量大的时候，导致其他queue的数据量也很大（因为每个queue中都有一份数据）。

（2）Kafka的高可用性：

使用的是Leader和follower的机制，只会从leader去读或写。

# 如何保证消息不被重复消费（如何保证消息消费时的幂等性）？

其实消息队列本身是不能阻止重复消费的情况的，目的不是去做到消息队列出现重复消费，而是去处理当出现重复消费的时候系统也能正常处理或者运行。